12_18 作業系統小考筆記

Tags: 小考筆記
Created: December 16 2017 +00:00, 02:37:50 Modified: December 17 2017 +00:00, 07:20:07
Note URL:
https://www.evernote.com/Home.action#n=950663f3-92bd-4312-ab05-511dcd506d23&b=346647db-96e5-4f2f-b8e5-82828ea46ccd&ses=4&sh=1&sds=5

[CH6]
解決**critical-section**問題的方法:

solution 需滿足的三個條件:

- mutual exclusion:
- 任一時間點，只允許一個process進入critical section內活動
- progress: 同時滿足2個要件
- 不想進入 critical section 的process不可以阻礙其他process進入critical section，即不可參與進入critical section 的決策過程
- 必須在有限的時間從想進入critical section的process中，挑選其中一個process進入critical section,**隱含No Deadlock**
- bounded waiting 自process提出 進入critical section的申請到獲准進入critical section的等待時間是有限的。即若有n個processes想進入，則任一process 至多等待n-1次即可進入，**隱含No starvation**

Peterson's Solution-軟體方式解決

```
do{
   flag[i] = TRUE;
   turn = j ; // 禮讓的概念

   while(flag[j]&&turn==j); // 想進去且輪到他進去
   CRITICAL SECTION
   flag[i] = FALSE;
   REMAINDER SECTION
} while(TRUE);
```

The structure of process Pi in Peterson's solution

假設有兩個process,共享兩個變數

[data structure]
int turn (代表輪到誰進入critical section)
ex:
turn==i process Pi is allowed to execute in its critical section

boolean flag[2] (代表誰準備好進入critical section)
ex:
flag[i] is true Pi is ready to enter its critical section

1 Mutual exclusion
若Pi與Pj皆想進入自己的Critical Section，代表 flag[i] == flag[j] == true，且分別執行到turn=i及turn=j 之設定，先後順序不同，turn的 僅會是i或j, 不會兩者皆是
2 Progress
若Pi 不想進Critical Section ,則表示flag[i] = false。此時若Pj想進入自己的Critical Section,必可通過
while(flag[i]andturn==i)do no-op這個空迴圈而進入CS，不會被Pi阻礙。
3 Bound-waiting
Pi 離開CS 後又企圖立刻進入自己的CS,此時Pi一定會執行turn=j,使得Pi無法再搶先於Pj進入自己的CS。所以Pj至多等待一次即可進入CS。

[hardware solution]
使用硬體 就不會有"synchronization"的問題

=>因為不會被中斷

Atomic (不被中斷) : TestAndSet()

```
boolean TestAndSet(bool &lock)
{
    bool value = lock;
    lock = TRUE;
    return value;
}
```

解說: excute atomically
return the value of "lock"
and set "lock" to TRUE

一開始 初始lock為false(0) , 第一個執行TestAndSet() 的process 會傳回 false,因此進入**critical section** ,在呼叫TestAndSet的同時 會將lock設成1 ,使得其他process無法進入， 當做完critical section,lock設成0,讓其他process也有機會進入critical section

"3條件" 符合狀況
mutual exclusion ? Yes
Progress ? Yes
Bounded-Wait ? No!
Why not?
因為是用搶的 看誰先call TestAndSet

mutex(mutual exclusion) locks
acquire(): acquires the lock
release(): releases the lock

```
acquire(){
    while(!available)
    ; /busy wait/
     available = false;
}
```

```
release{
    available = true;
}
```

缺點: requires busy waiting -> wastes CPU cycles
real multiprogramming system,where a single CPU is shared among many processes
優點: no context switch(耗時) is required when a process must wait on a lock
good for multiprocessor system

Semaphore
A tool to generalize the synchronization problem.
easy to solve , but no guarantee for correctness

a record of how many units of a particular resources are available

if #record = 1 -> binary semaphore,mutex lock
if #record > 1 -> counting semaphore

classical problems of synchronization

purpose:用來驗證解決synchronization的解法有沒正確

- Bounded-Buffer (Producer-Consumer) Problem
- Reader-Writers Problem (檔案，資料的操作)
- Dining-Philosopher Problem

Bounded-Buffer Problem
buffer:
空的時候 => consumer等
滿的時候 => producer等
[CH5]